Temat projektu

Mamy na płaszczyźnie XY dokładnie N punktów. W jednym ruchu możemy zaznaczyć zbiór punktów, z których wszystkie są współliniowe, i usunąć je. Celem gry jest usunięcie wszystkich punktów w jak najmniejszej liczbie ruchów.

Zaproponuj algorytm, który wyliczy minimalną liczbę ruchów potrzebną do usunięcia wszystkich punktów oraz liczbę możliwych kombinacji ruchów służących usunięciu tych punktów (przy czym jeden ruch różni się od innego jeśli przynajmniej jeden punkt jest usunięty w innym ruchu).

Struktury danych

Line oraz Point

Reprezentują linie i punkty. Każda linia trzyma listę punktów, które ta linia zawiera. Podobnie każdy punkt trzyma listę linii, które przecinają ten punkt.

Dokładniej ujmując to obie te struktury trzymają iteratory na tą drugą, co jest możliwe ponieważ punkty oraz linie są trzymane w liście, której iteratory są stałe.

Dzięki użyciu iteratorów nie trzeba korzystać z pointerów, ani kopiować obiektów.

PlaneXY

Trzyma listę punktów oraz wektor list które trzymają linie. Każdy indeks wektora odpowiada rozmiarowi danej linii (czyli ilości przecinanej przez nią punktów).

Dzięki temu linie są zawsze posortowane według rozmiaru, co okazuje się przydatne w trakcie działania algorytmu 2 i 3.

Dostępne są metody które "niby wyłączają" daną linie - tzn. symulują układ struktury danych tak jakby linia i wszystkie punkty które ona zawiera została usunięta.

Możliwy jest także "splice" danych linii, co jest używane, aby przesunąć przetestowane już linie do innej struktury danych. Dzięki temu dana kombinacja linii jest sprawdzana max raz.

Algorytmy rozwiązujące problem

Problem został podzielony na 3 zasadnicze podproblemy:

1. Znalezienie wszystkich linii przecinających 3 lub więcej punktów, dla znanego zbioru punktów
2. Znalezienie minimalnej liczby linii, które przecinają wszystkie punkty, znając zbiór linii
3. Znalezienie liczby możliwych kombinacji linii, które przecinają wszystkie punkty, znając liczbę linii których trzeba użyć

Algorytm numer 1 - LineGen

Aby móc rozważać, które linie należy wybrać, aby spełnić warunki polecenia, trzeba najpierw te linie określić.

W tym celu wybieram ze zbioru punktów dwa punkty, określam na jakiej linii te dwa punkty leżą i sprawdzam czy linia o takich parametrach nie została już rozważona. (to ostanie realizowane za pomocą unordered_set)

Jeśli taka linia rzeczywiście została już rozważona to wracam do początku algorytmu wybierając dwa kolejne punkty.

Następnie dla wszystkich punktów za wyjątkiem tych już rozważonych, sprawdzam czy należą do rozważanej linii. Jeśli tak to dodaje punkt do linii.

Po przejrzeniu wszystkich punktów sprawdzam, czy w rozważanej linii leżą więcej niż 2 punkty. Jeśli tak to taką linię zapisuję i wracam do początku algorytmu.

Postępuję tak dopóki nie rozważę wszystkich par punktów.

Algorytm numer 2 - MinCover

Przed właściwym algorytmem znajdowania minimalnego pokrycia, następuje preprocesing.

Znajduje on punkty które nie leża na żadnej z linii znalezionych za pomocą algorytmu 1, i odejmuje znalezioną liczbę od liczby punktów dla których szukane będzie rozwiązanie.

Preprocesing znajduje także linie, które mają co najmniej 2 punkty które są przecinane tylko przez tą linie. Taka linia może zostać usunięta z danych wejściowych do właściwego algorytmu.

Po tych zabiegach następuje przejście do właściwej części algorytmu.

Algorytm przyjmuje liczbę punktów (n), przypuszczalną liczbę linii (k), oraz zbiór linii wygenerowany algorytmem z pkt. 1 (i poddany preprocesingowi).

Liczba punktów jest dana, natomiast jako parametr początkowy k podawana jest liczba ceil(n/2) - czyli tak jakbyśmy nie przewidywali żadnych linii które zawierają więcej niż 2 punkty.

Dla każdej linii która przecina więcej niż ceil(c/k) punktów, po zasymulowaniu usunięcia tejże linii i przerzucania jej do listy przetestowanych, wywoływany jest rekurencyjnie ten algorytm.

Gdy rekurencja wraca, przetestowana linia jest "włączana" (ale cały czas pozostaje w liście przetestownych) i wybierana jest kolejna linia.

Kiedy znalezione zostaje rozwiązanie dla k linii, parametr k zostaje dekrementowany, a nowe stany są rozwijane z ostrzejszym ograniczeniem.

Gdy nie ma już stanów które możnaby rozwinąć zwracane jest najlepsze znalezione dotychczas rozwiązanie, a jeśli takiego nie ma -1, natomiast lista przetestowanych linii jest włączana do głównego zbioru, aby zachować jego spójność.

Algorytm numer 3 - UniqueSolutions

Pomysł na algorytm do problemu numer 3 jest modyfikacją algorytmu z punktu 2.

Podobnie przeszukujemy w głąb bez linii które nie zawierają więcej niż n/k punktów.

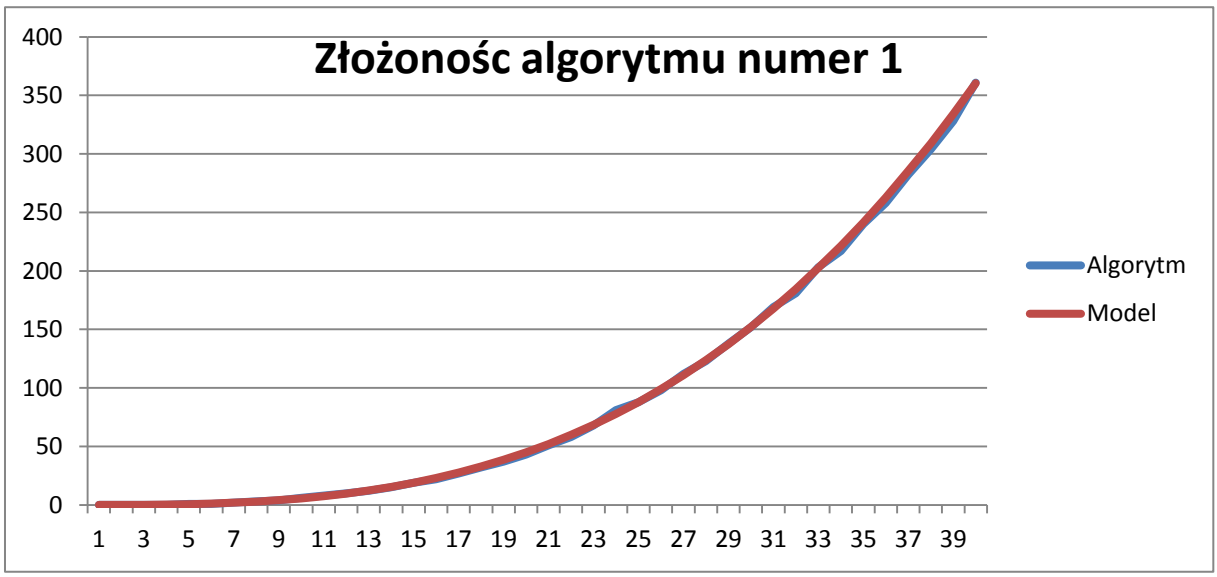
Zmianą jest to, że parametr k jest stały (wyliczony za pomocą algorytmu numer 2), a zwracane jest nie nowe rozwiązanie, a liczba znalezionych unikalnych rozwiązań.

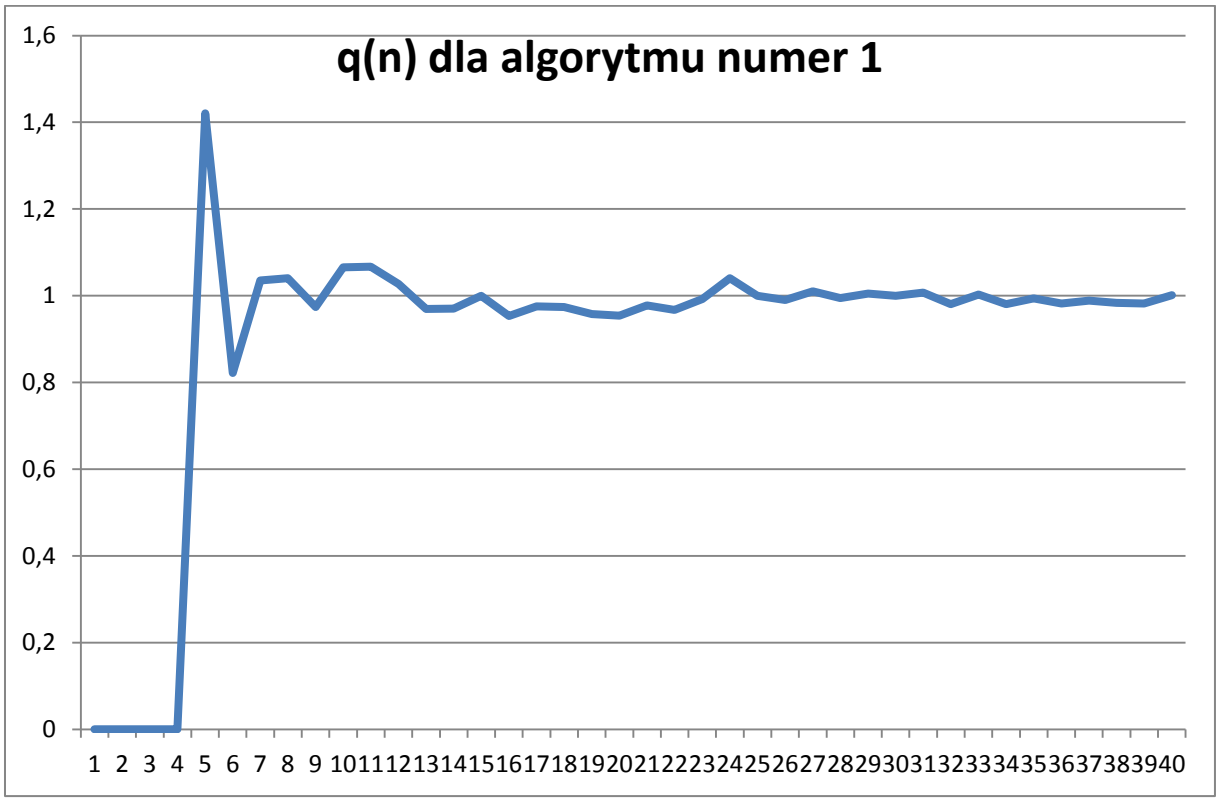
Ocena złożoności algorytmów

Algorytm numer 1 - LineGen

Algorytm numer 1 jest wykonywany dla każdej pary punktów, a więc $\frac{1}{2}n^2$ razy. Natomiast podczas rozważania każdej pary punktów sprawdzane jest średnio $\frac{1}{2}n$ innych punktów. Daje to $T(n) = \frac{1}{4}n^3$.

Tak więc złożoność algorytmu dla generacji linii wynosi: $\boldsymbol{O(n^3)}$.





Algorytm numer 2 i 3 – MinCover, UniqueSolutions

Najgorszym teoretycznym przypadkiem dla obu algorytmów, jest taki zbiór punktów i linii, dla których zmienna limit, nie jest w stanie obciąć niczego w drzewie.

Liczba stanów które należałoby rozwinąć w takim przypadku jest rozwiązaniem rekurencji: $B(n,k) = B(n-1,\ k-1) + B(n-2,\ k-1) + \ldots + B(1,\ k-1)$

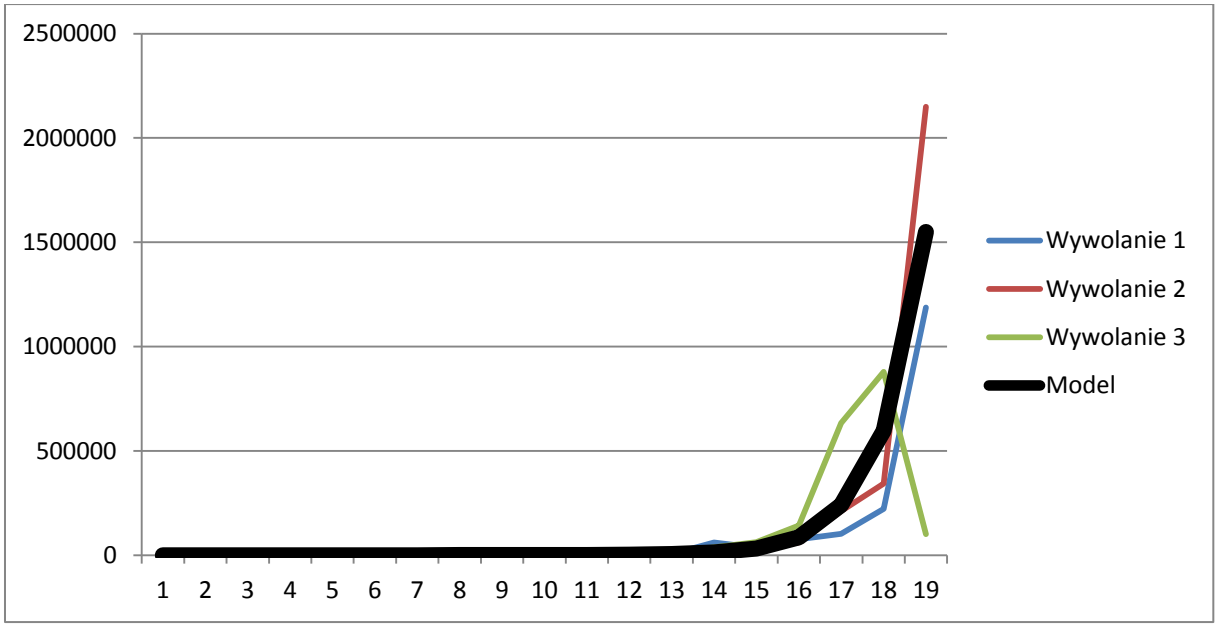
Okazuje się że rozwiązaniem takiej rekurencji jest $\sum_{i=0}^{k}\binom{l}{i}$, gdzie l to liczba wygenerowanych linii, a k to najmniejsze pokrycie danego zbioru punktów.

Niestety wygenerowanie takiego teoretycznie najgorszego przypadku, mimo wielu prób, okazało się niemożliwe ponieważ nawet małe zmiany w zbiorze wygenerowanych linii, mogły powodować znaczne zmiany ilości obcinanych gałęzi drzewa rozwiązań, co ilustruje poniższy wykres, gdzie na określonej powierzchni stopniowo dodawano kolejne punkty.

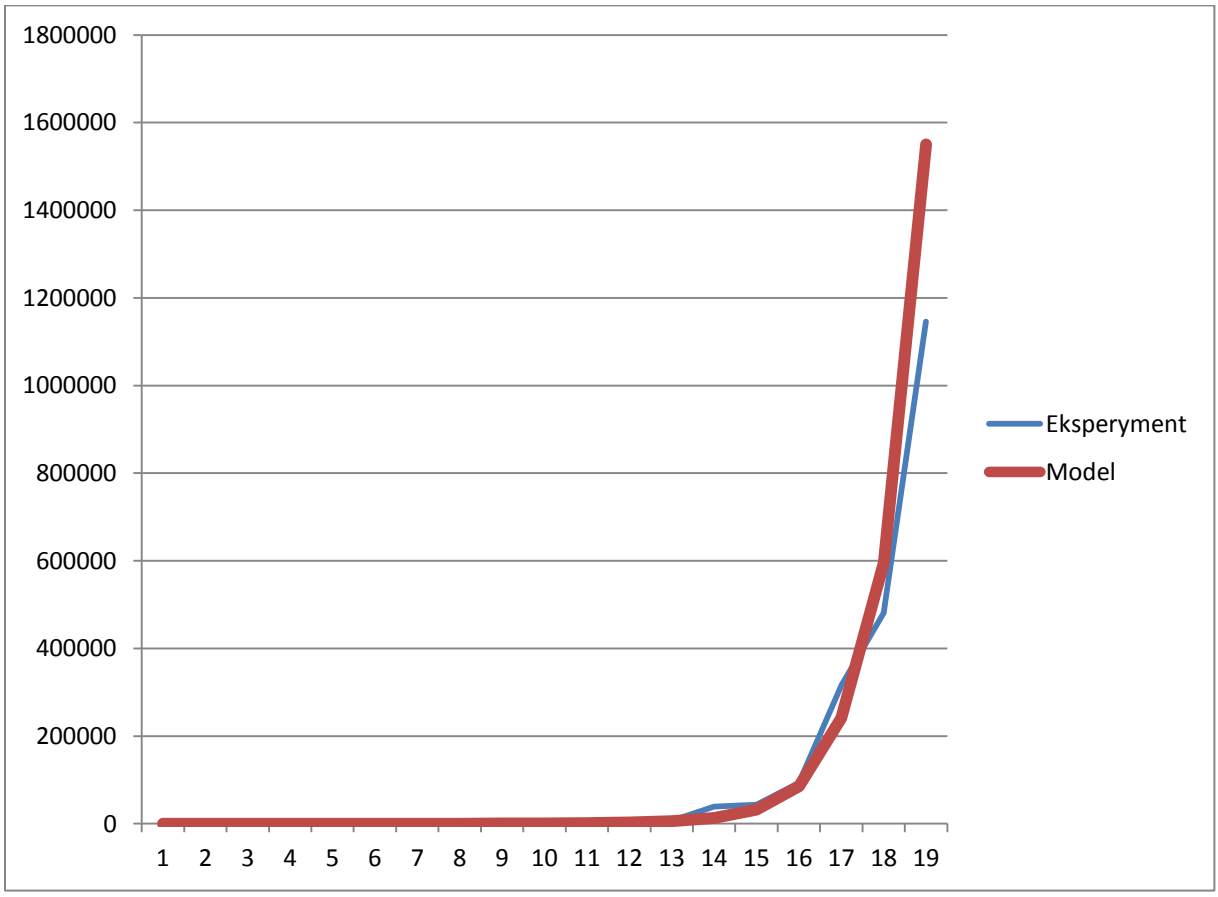
Ponieważ eksperyment w przypadku zmiennej gęstości punktów (powyższy wykres) nie dawał szans na zamodelowanie, postanowiłem zmienić podejście – zmienić sposób działania eksperymentów, oraz wyznaczyć złożoność obu algorytmów eksperymentalnie.

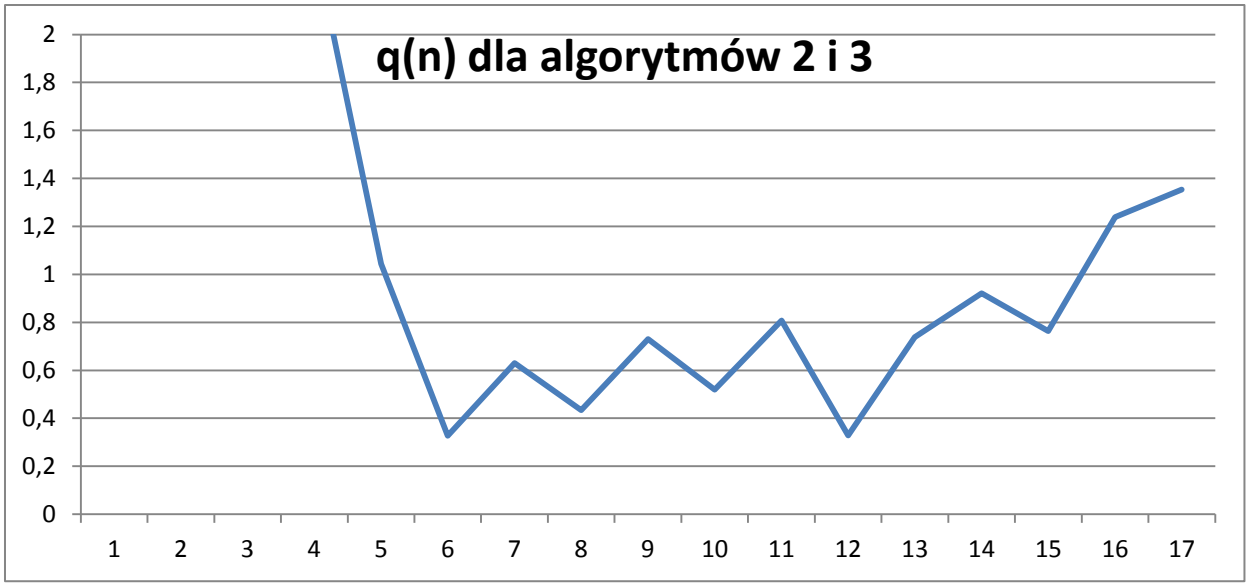
Po wielu eksperymentach, okazało się że najlepiej oba algorytmy modeluje złożoność: $\boldsymbol{O}(\mathbf{2^k})$

Postanowiłem wykonywać eksperymenty w oparciu o stałą gęstość punktów, a zmienną powierzchnie, co dawało dużo lepsze, wyniki eksperymentów:



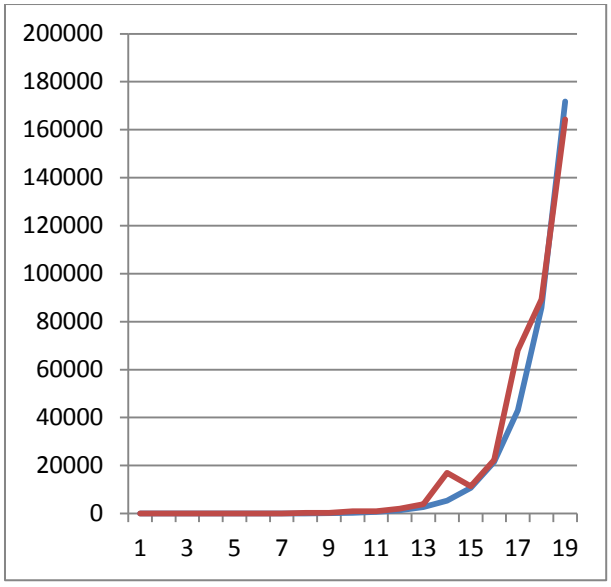
Mimo to, jak widać na wykresie poszczególne wywołania eksperymentów dawały dosyć zmienne wyniki, przez co dokładne zamodelowanie okazało się bardzo problematyczne, nawet po uśrednieniu wyników wywołań:

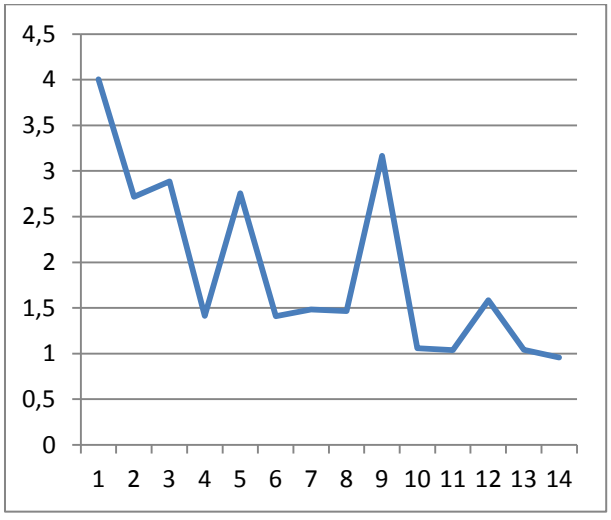




Inne przykłady:

Algorytm 2





Algorytm 3